# Как сделать исполнение low-code прозрачным: опыт большой IoT-платформы

Владимир Плизга
Tibbo Systems


Saint HighLoad++

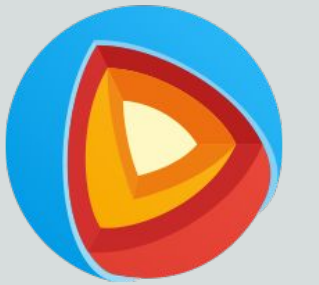
# ПРИВЕТ!

Я – Владимир Плизга́

- В коммерческой разработке с 2011 г.
  - Строил интернет-банкинг в финтехе
- Развиваю платформу AggreGate с 2021 г.

# LOW-CODE-ПЛАТФОРМА AGGREGATE

- Используется для создания приложений на базе Интернета вещей (**IoT**)
- Представляет собой набор “кирпичиков” под разные нужды:
  - **50+** модулей для интеграции с устройствами
  - **≈70** модулей для обработки, передачи, визуализации и хранения данных с них
- Чаще всего применяется в IIoT (Industrial IoT)



AggreGate

# РЕШИТЕ КАПЧУ

Выберите объекты, относящиеся к интернету вещей



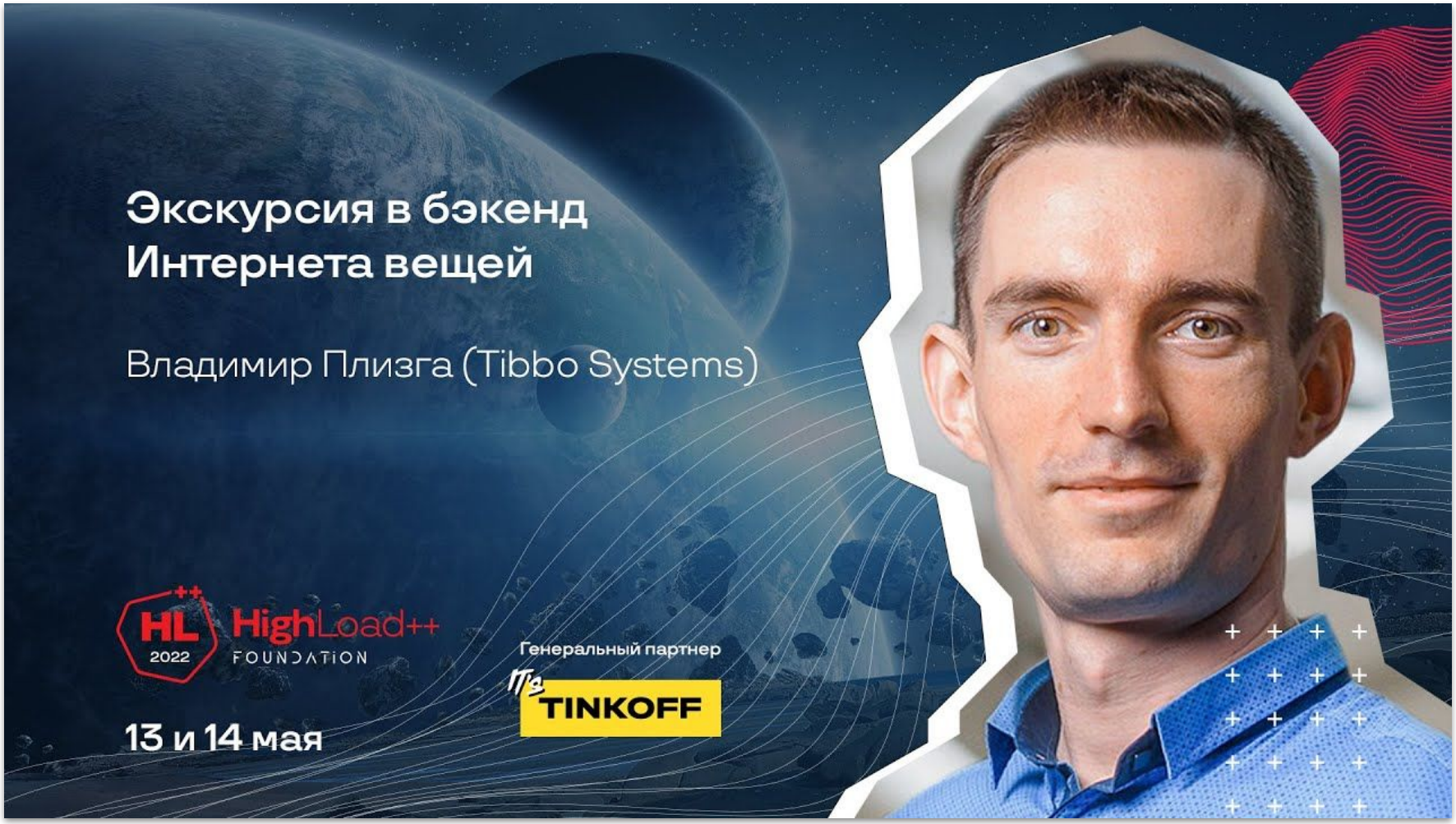


# РЕШИТЕ КАПЧУ

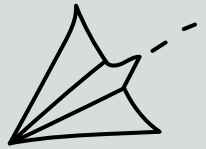
Выберите объекты, относящиеся к интернету вещей

# ПРИЧЕМ ЗДЕСЬ КОРОВЫ?

Слайды и видео доклада



# В ЧЕМ РАЗНИЦА?



- Customer IoT (CIoT) – **домашний** интернет вещей
  - Бытовые устройства (“умный дом”)
- Industrial IoT (IIoT) – **промышленный** интернет вещей
  - Устройства на производстве, транспорте, в полях, …
  - И их может быть **много тысяч** 😱

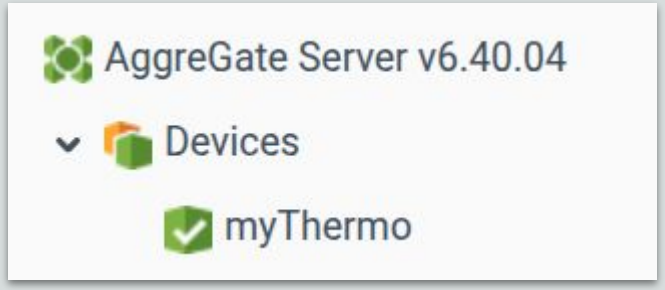
# РАЗБЕРЕМ НА ПРИМЕРЕ

## САМОДЕЛЬНАЯ МЕТЕОСТАНЦИЯ

- **Задачи**:
  - сделать собственный UI в браузере
  - собирать аналитику без программирования
- **Основа**: “умный” термометр и AggreGate

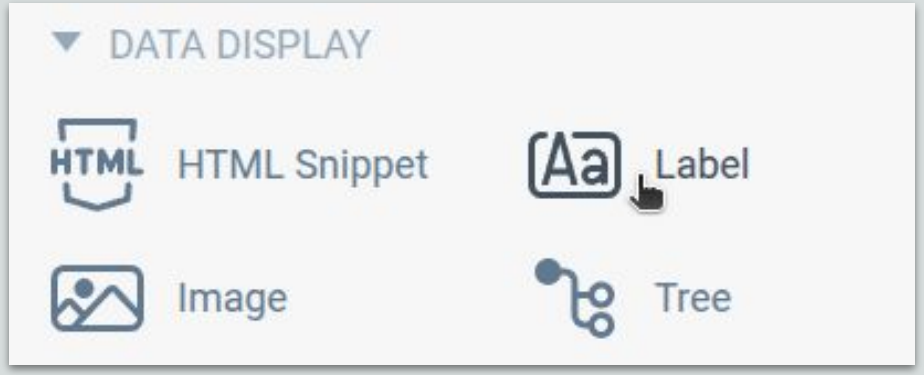
# ОСНОВНЫЕ ПОНЯТИЯ 1/3

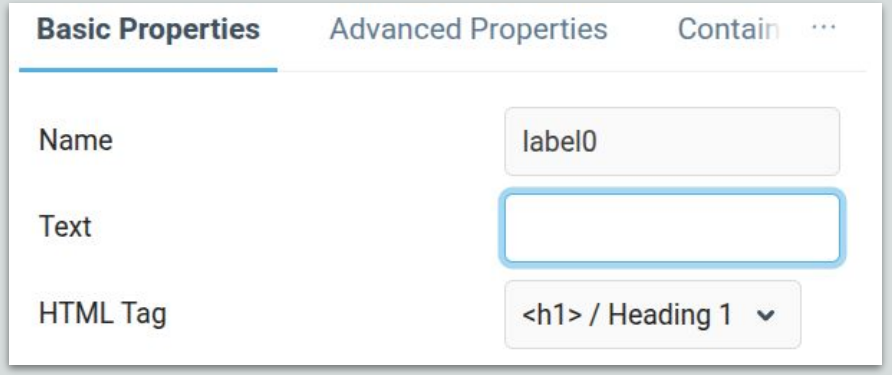
- Главные “кирпичики” в AggreGate – контексты
  - Контекст термометра: **`users.admin.devices.myThermo`**
- У контекстов бывают функции, события и переменные
  - Температура доступна в виде переменной **`temperature`**



# ОСНОВНЫЕ ПОНЯТИЯ 2/3

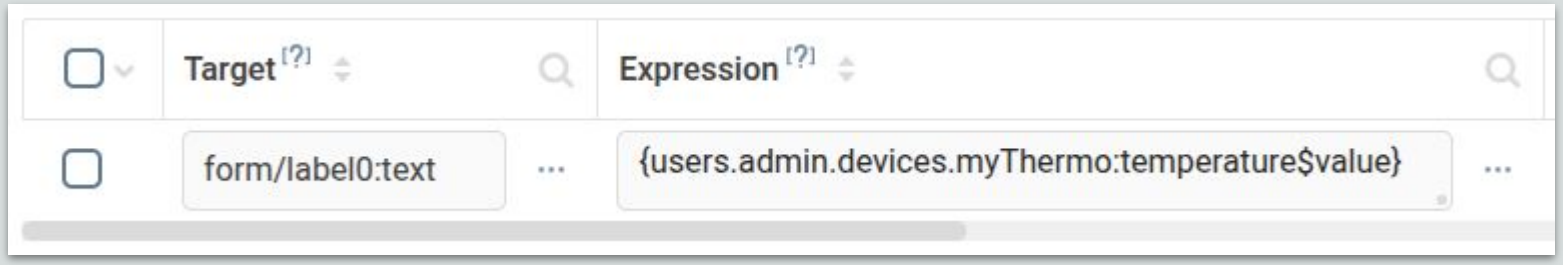
- Визуальные компоненты дашбордов – тоже контексты
  - Но они живут только в дашбордах
- У компонентов тоже есть переменные
  - Например, у компонента Label есть переменные **`name`** и **`text`**



# ОСНОВНЫЕ ПОНЯТИЯ 3/3

- Для связывания данных между контекстами есть привязки
  - Описывают, где брать данные, как менять и куда записывать
- По смыслу похожи на формулы в Excel
- Пишутся на Языке выражений AggreGate
  - Собственный интерпретируемый язык платформы



# АЗЫ ССЫЛОК И ВЫРАЖЕНИЙ

**“Голая” ссылка**

```
{users.admin.devices.myThermo:temperature$value}
```

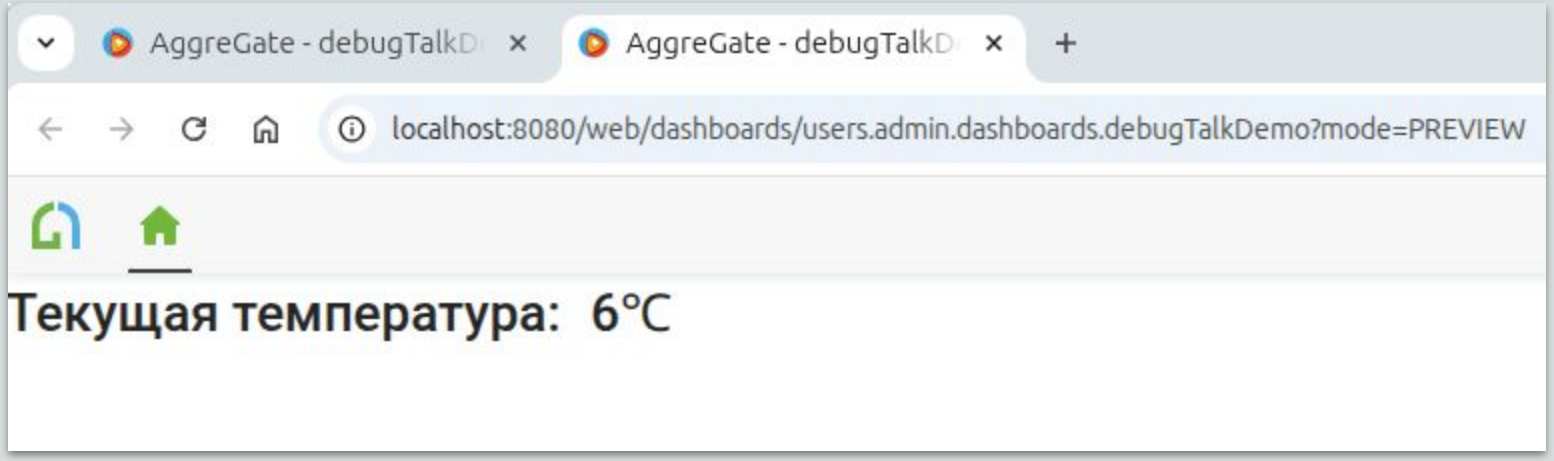
**Конвертация из Фаренгейта в Цельсия**

```
5/9 * ({users.admin.devices.myThermo:temperature$value} - 32)
```

**Придание читабельности**

```
round(5/9 * ({users.admin.devices.myThermo:temperature$value} - 32)) + "℃"
```

# ДЕМО РАБОТАЮЩЕГО ДАШБОРДА



# ДОБАВЛЯЕМ АНАЛИТИКУ

| Условие для напоминания о переобувке машины |
| --- |
| |

# ДОБАВЛЯЕМ АНАЛИТИКУ

**Условие для напоминания о переобувке машины**

```
"users.admin.devices.myThermo",
"temperature",
```

# ДОБАВЛЯЕМ АНАЛИТИКУ

**Условие для напоминания о переобувке машины**

```
callFunction(
  "utilities",
  "variableHistory",
  "users.admin.devices.myThermo",
  "temperature",
```

# ДОБАВЛЯЕМ АНАЛИТИКУ

**Условие для напоминания о переобувке машины**

```
callFunction(
  "utilities",
  "variableHistory",
  "users.admin.devices.myThermo",
  "temperature",
  dateAdd(now(), -1, "day"),
  now()
),
```

# ДОБАВЛЯЕМ АНАЛИТИКУ

**Условие для напоминания о переобувке машины**

```
avg(
 callFunction(
   "utilities",
   "variableHistory",
   "users.admin.devices.myThermo",
   "temperature",
   dateAdd(now(), -1, "day"),
   now()
 ),
 "value"
) < 5.0
```

# ПРОМЕЖУТОЧНЫЙ ИТОГ

**Что имеем:**

- простенькое low-code-приложение
- 3 контекста, 1 привязка и 2 выражения

**Чего хотим:**

- сбор, аналитику, визуализацию с других устройств
- расчет температуры по ощущениям (“RealFeel”)
- прогнозы на основе ML

# К ЧЕМУ ГОТОВИТЬСЯ ДАЛЬШЕ

**Пример 1**

```
{0} == 0
? 1
: {0} *
 cell(
   callFunction(
     {.:}
     , "factorial"
     , {0} - 1
   )
 )
```

## Пример 2

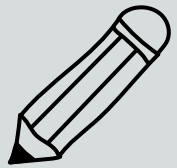
```
aggregate(
   {users.admin.models.elementsModel:elementsProperties}
   , "addRecords({env/previous}, {elementKey}, "+
      "aggregate("+
         "{properties}"+
     ", 'records(filter(dt(), \"{tabDesc} == \\\'\"+{tabDesc}+\"\\\'\")) > 1 &&
!hasField({propertyTable}, \"propertyTable\")"+
     "? {env/previous}"+
         ": addRecords({env/previous}, {tabDesc},{propertyTable}, {name})'"+
         ", table('<<tabDesc><S><A⟹<<propertyTable><T><A=<F=>>><<name><S><A⟹')"+
      ")"
   +")"
   , table("<<elementKey><S><A⟹<<properties><T><A=<F=>>>")
)
```

# Пример 3

```
aggregate(
 {users.admin.models.elementsModel:elementsProperties}
 , "addRecords({env/previous}, {elementKey}, "+
   "aggregate("+
     "{properties}"+
     ", 'addRecords({env/previous}, {tabDesc},'+"+
      "'hasField({propertyTable}, \"propertyTable\")'+"+
        "'? aggregate({propertyTable}, \"addRecords({env/previous}, {tab}, '+"+
          "'describe({propertyTable}'+"+
            "', \\\'visible\\\', \\\'Видимый\\\''+"+
            "', \\\'externalButtons\\\', \\\'Кнопки\\\''+"+
            "', \\\'stepsCount\\\', \\\'Количество шагов\\\''+"+
            "', \\\'slideCount\\\', \\\'Количество слайдов\\\''+"+
            "', \\\'placement\\\', \\\'Расположение выдвижной панели\\\''+"+
            "', \\\'size\\\', \\\'Размер выдвижной панели\\\''+"+
            "', \\\'mask\\\', \\\'Маска выдвижной панели\\\''+"+
            "', \\\'buttonClasses\\\', \\\'Пользовательские классы кнопки\\\''+"+
            "', \\\'expression\\\', \\\'Выражение выбора строк\\\''+"+
            "', \\\'columnName\\\', \\\'Имя столбца идентификатора записи\\\''+"+
            "', \\\'inputPlaceholder\\\', \\\'Заполнитель в фильтре\\\''+"+
            "', \\\'keywordFilter\\\', \\\'Фильтр ключевых слов\\\''+"+ // Context List → Advanced → Search
            "', \\\'imageButtonContainerStyle\\\', \\\'Контейнер изображения\\\''+"+ // Button → Advanced → Style
            "', \\\'showSearch\\\', \\\'Показать строку поиска\\\''+"+ // Dropdown Tree → Advanced → Search
            "', \\\'sliderType\\\', \\\'Вид регулятора\\\''+"+ // Slider → Advanced → Component
            "', \\\'currentOptions\\\', \\\'Выбранные опции\\\''+"+ // Dropdown List → Advanced → Component
            "', \\\'polylinesStyles\\\', \\\'Track Styles\\\''+"+ // Map → Advanced → Style
            "', \\\'areasStyles\\\', \\\'Area Styles\\\''+"+ // Map → Advanced → Style
            "', \\\'circleAreasStyles\\\', \\\'Circular Area Styles\\\''+"+ // Map → Advanced → Style
          "')'+"+
          "')\", table(\"<<tab><S><F=N><A=<NULL >>><<propertyTable><T><A=<F= >>>\"))'+"+
        "': {propertyTable}'+"+
     "', {name})'"+
     ", table('<<tabDesc><S><A⟹<<propertyTable><T><A=<F= >>><<name><S><A⟹')"+
   ")"
 +")"
 , table("<<elementKey><S><A⟹<<properties><T><A=<F= >>>")
)
```

# СООТНОШЕНИЕ МАСШТАБОВ IOT

| Характеристика | Customer IoT | Industrial IoT | Примечание |
|---|---|---|---|
| Число контекстов | ≲50 | ≲50K* | * С учетом автогенерённых |
| Число выражений | ≲50 | ≲1000* | * Не считая автогенерённых |
| Частота поступления обновлений | ≲1 RPS | ≲500 KRPS* | * За счёт множества устройств |
| Число пользователей | ≲5 | ≲1000* | * В отдельных случаях до 50K |
| Основные акценты | Наглядность, простота, экономность | Производительность, безопасность, отказоустойчивость | |



# ПРОБЛЕМЫ МАСШТАБОВ IIOT

**Сложно читать**

… навороченные выражения, причем как коллег, так и свои собственные

**Трудно отлаживать**

… выражения при создании и доработках:

- много зависимостей
- риски side-эффектов
- когнитивная сложность

**Тяжело разбираться**

… в приложении в целом, т.к. связи между его компонентами неочевидны

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

**Сложно читать**

**Трудно отлаживать**

**Тяжело разбираться**

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

**Сложно читать**

**Деревья разбора**

Чтобы “познавать” выражение от общего к частному

**Трудно отлаживать**

**Тяжело разбираться**

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

| Сложно читать | Трудно отлаживать | Тяжело разбираться |
| --- | --- | --- |
| **Деревья разбора** | **Трассировка вычислений** | |
| Чтобы “познавать” выражение от общего к частному | Чтобы каждый шаг исполнения был виден в дереве | |

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

| Сложно читать | Трудно отлаживать | Тяжело разбираться |
| --- | --- | --- |
| **Деревья разбора** | **Трассировка вычислений** | **Граф взаимодействий** |
| Чтобы “познавать” выражение от общего к частному | Чтобы каждый шаг исполнения был виден в дереве | Чтобы картина обращений между контекстами стала явной |

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

| Сложно читать | Трудно отлаживать | Тяжело разбираться |
|---|---|---|
| **Деревья разбора** | **Трассировка вычислений** | **Граф взаимодействий** |
| Чтобы “познавать” выражение от общего к частному | Чтобы каждый шаг исполнения был виден в дереве | Чтобы картина обращений между контекстами стала явной |
| 1 | 2 | 3 |

1

# ДЕРЕВЬЯ РАЗБОРА ДЛЯ НАГЛЯДНОСТИ

# ЧТО ЭТО ЗА ДЕРЕВЬЯ

- Результат парсинга выражения
- В виде разновидности **ориентированного** графа – дерева
- Узлы – элементы выражения: константы, операторы, ссылки, …
  - Корневой узел – результат всего выражения
- Связи – отношения вхождения одних элементов в другие

# ЧЕМ ПАРСИТЬ ВЫРАЖЕНИЯ?

- Для Java есть 2 основных варианта:
  - **JavaCC** (и его клоны)
  - **ANTLR4**
- **JavaCC** старый, ущербный, полузаброшен
- **ANTLR** удобен, документирован, развивается
- Мы выбрали JavaCC

*Парсинг грамматики Kotlin*

| **Парсер** | **1-ый запуск** | **1000-ый запуск** |
|---|---|---|
| ANTLR | 3705 мс | 137,2 мс |
| JavaCC | 19 мс | 0,1 мс |

https://habr.com/ru/articles/433000/

# КАК ВИЗУАЛИЗИРОВАТЬ?

- Конечная цель – интерактивный граф на фронте (ReactJS, Ant.Design)
- Но нужно видеть результат при разработке
- Вариант решения – экспорт данных в виде Graphviz DOT:
  - Текстовый человеко-читаемый формат
  - Соответствует структуре данных Ant.Design (2 списка)
  - Много инструментов визуализации, в т.ч. онлайн

# ПРОСТЕЙШИЙ ПРИМЕР

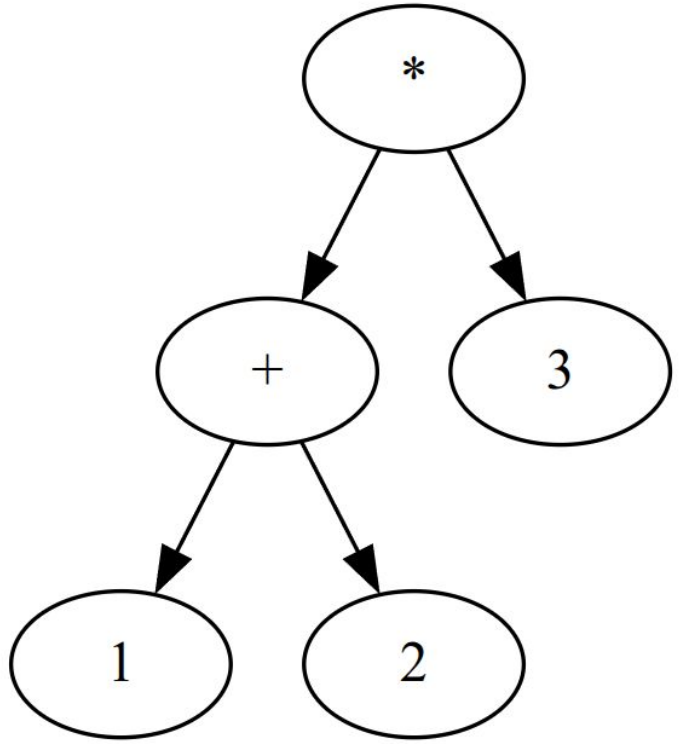
**Выражение**

```
(1 + 2) * 3
```

**Graphviz DOT**

```
digraph G {
  0[label="*"]
  1[label="+"]
  2[label="1"]
  3[label="2"]
  4[label="3"]
  0→1
  1→2
  1→3
  0→4
}
```

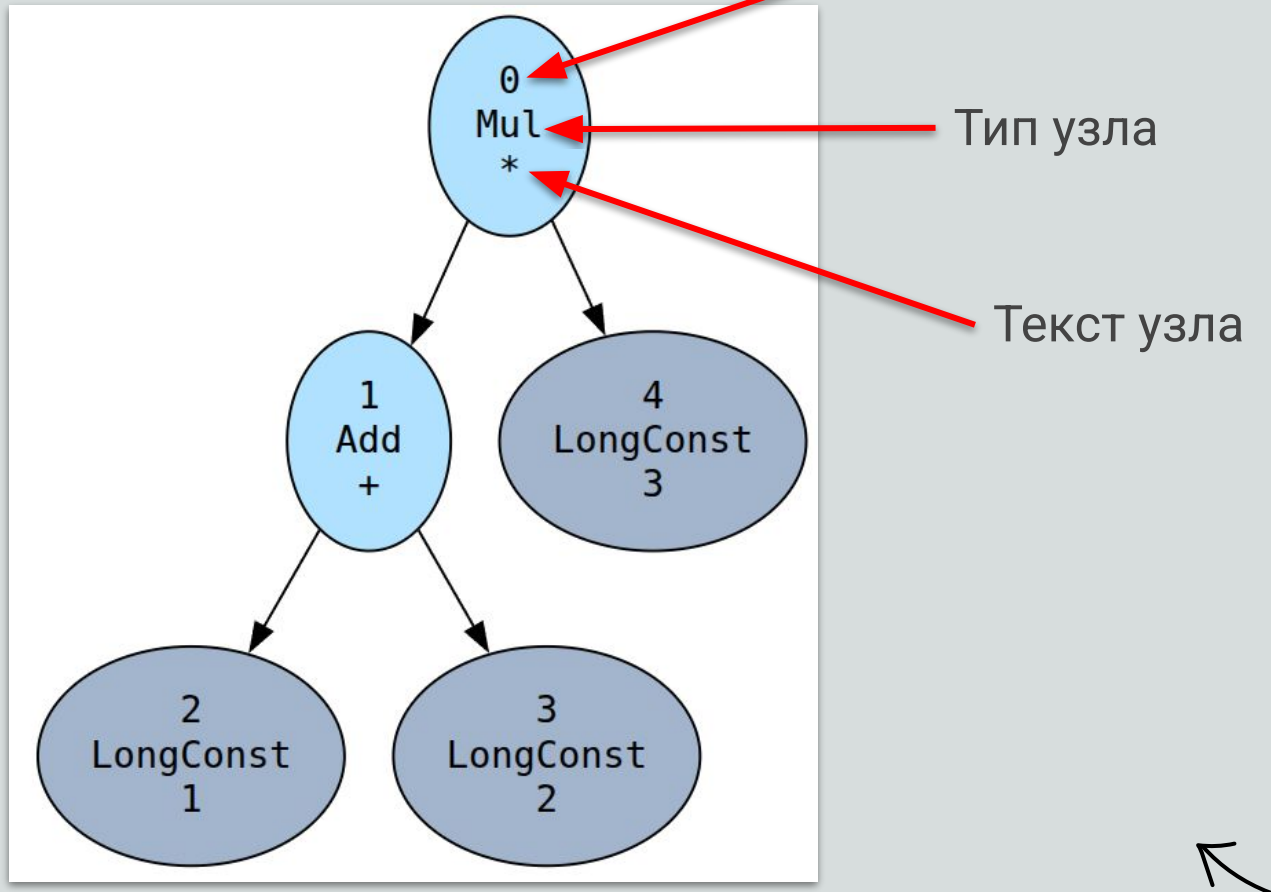
**Дерево разбора**



# ИНФОРМАТИВНЫЕ УЗЛЫ

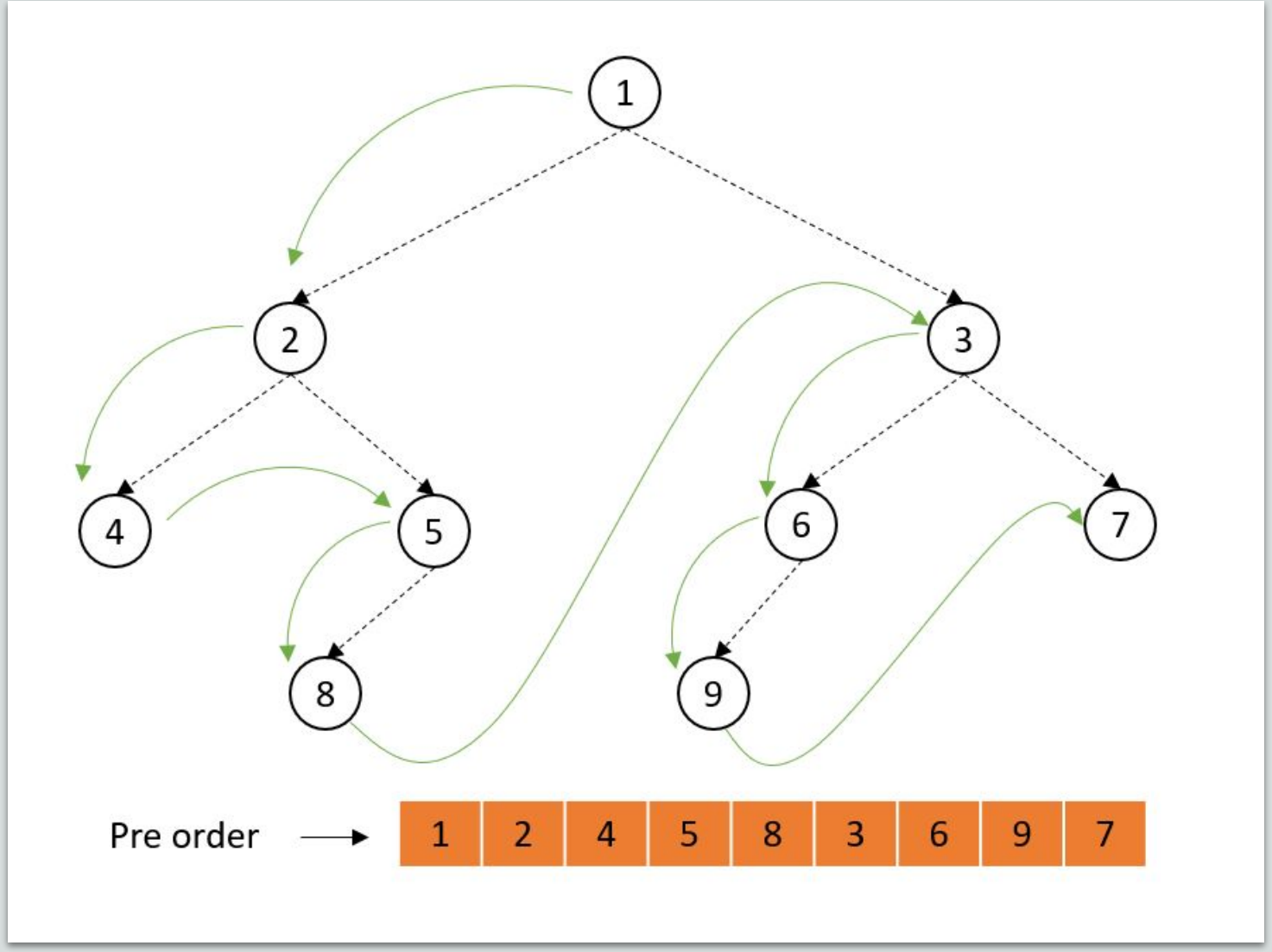
**Выражение**

```
(1 + 2) * 3
```



# ИНДЕКС ОБХОДА

Порядковый номер узла при леворекурсивном обходе дерева




https://python.plainenglish.io/iterative-tree-traversal-15c1b87d8d37

# ЧТО ЭТО ДАЕТ ПОЛЬЗОВАТЕЛЯМ

**Показывает**

, откуда начинать
читать выражение

# ЧТО ЭТО ДАЕТ ПОЛЬЗОВАТЕЛЯМ

**Показывает**

, откуда начинать
читать выражение

**Позволяет**

не вдаваться в детали
с самого начала

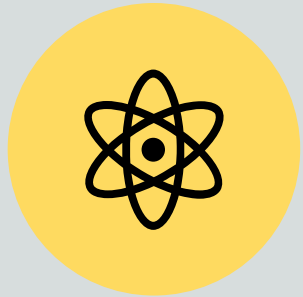
# ЧТО ЭТО ДАЕТ ПОЛЬЗОВАТЕЛЯМ

**Показывает**

, откуда начинать
читать выражение

**Позволяет**

не вдаваться в детали
с самого начала



**Создает**

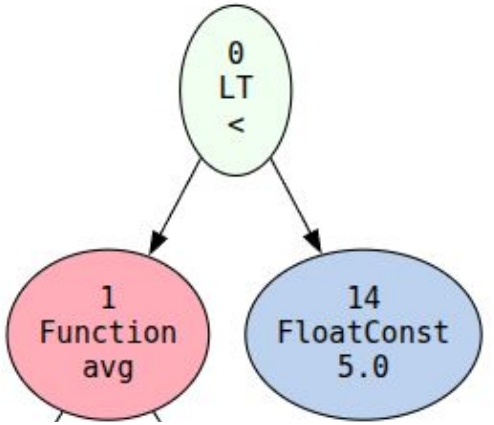
основу для
визуализации
вычислений

```
avg(
 callFunction(
   "utilities",
   "variableHistory",
   "users.admin.devices.myThermo",
   "temperature",
   dateAdd(now(), -1, "day"),
   now()
 ),
 "value"
) < 5.0
```

# НА ПРИМЕРЕ

выражения для напоминания
о переобувке машины

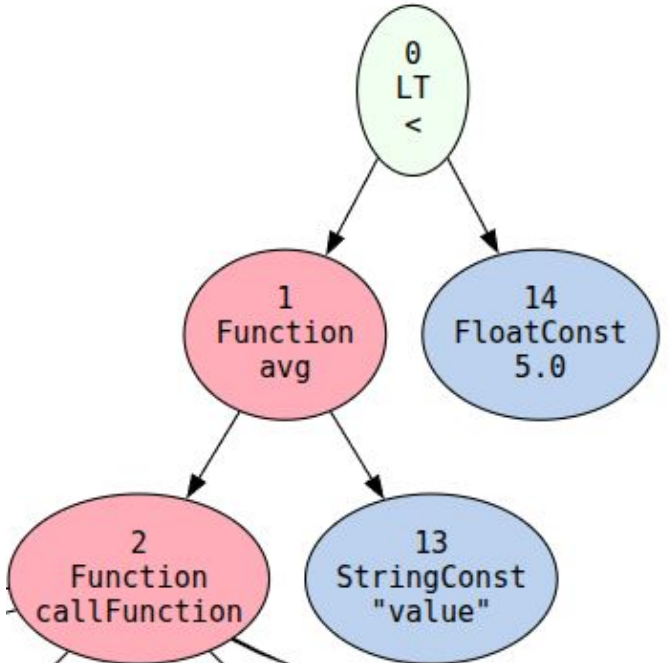
```
avg(
 callFunction(
   "utilities",
   "variableHistory",
   "users.admin.devices.myThermo",
   "temperature",
   dateAdd(now(), -1, "day"),
   now()
 ),
 "value"
) < 5.0
```




# НА ПРИМЕРЕ

выражения для напоминания
о переобувке машины

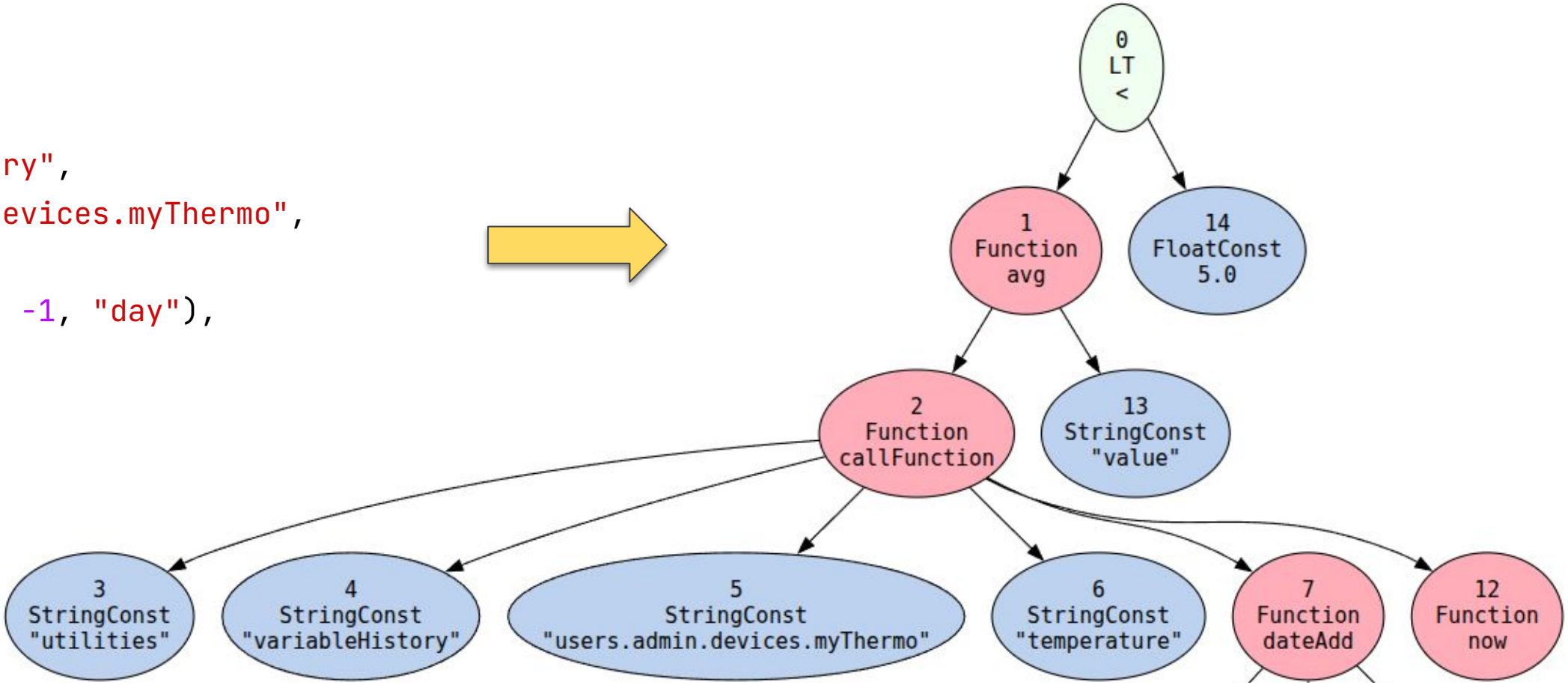
```
avg(
 callFunction(
   "utilities",
   "variableHistory",
   "users.admin.devices.myThermo",
   "temperature",
   dateAdd(now(), -1, "day"),
   now()
 ),
 "value"
) < 5.0
```




# НА ПРИМЕРЕ

выражения для напоминания
о переобувке машины

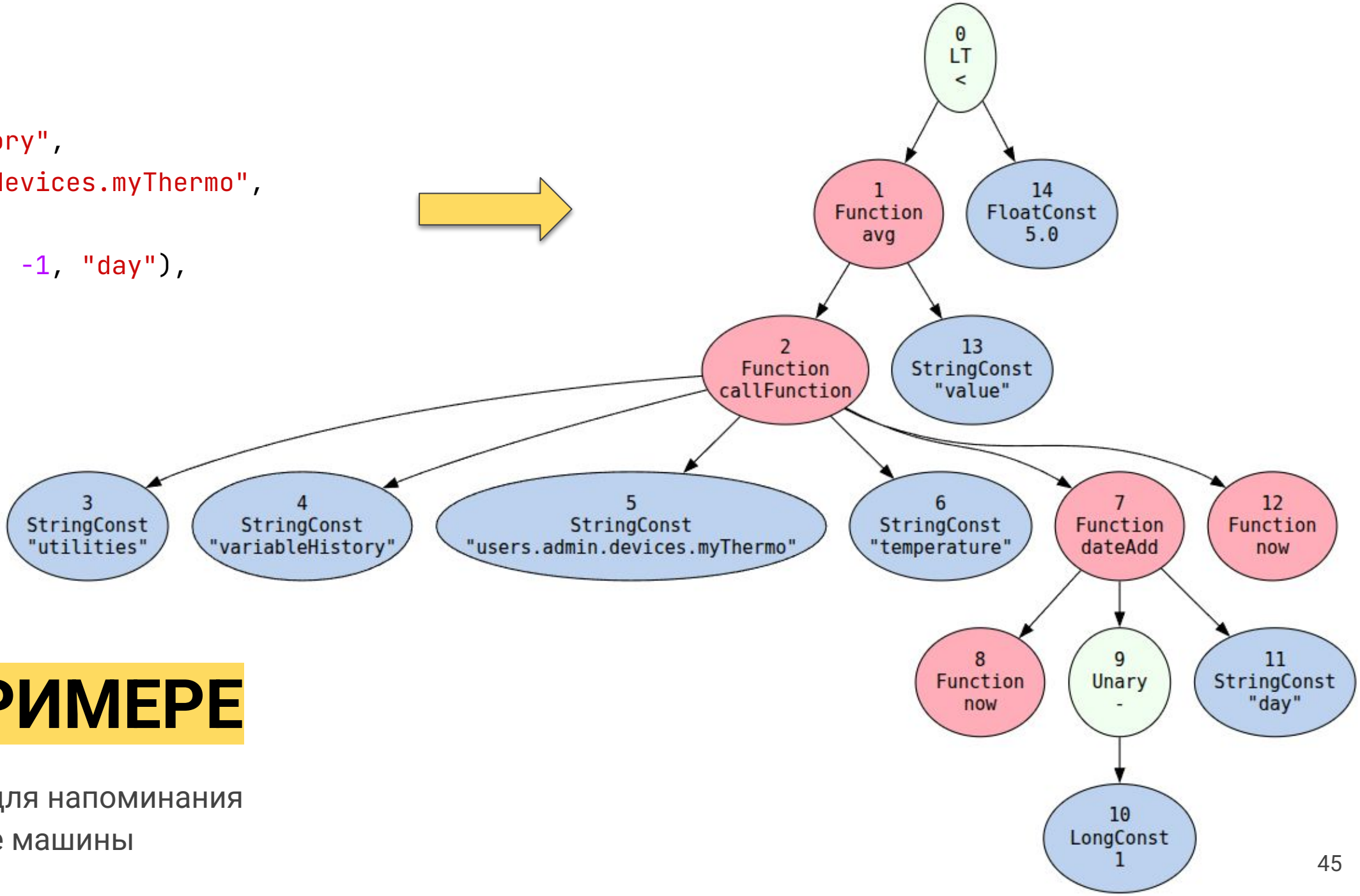
```
avg(
 callFunction(
   "utilities",
   "variableHistory",
   "users.admin.devices.myThermo",
   "temperature",
   dateAdd(now(), -1, "day"),
   now()
 ),
 "value"
) < 5.0
```




# НА ПРИМЕРЕ

выражения для напоминания
о переобувке машины

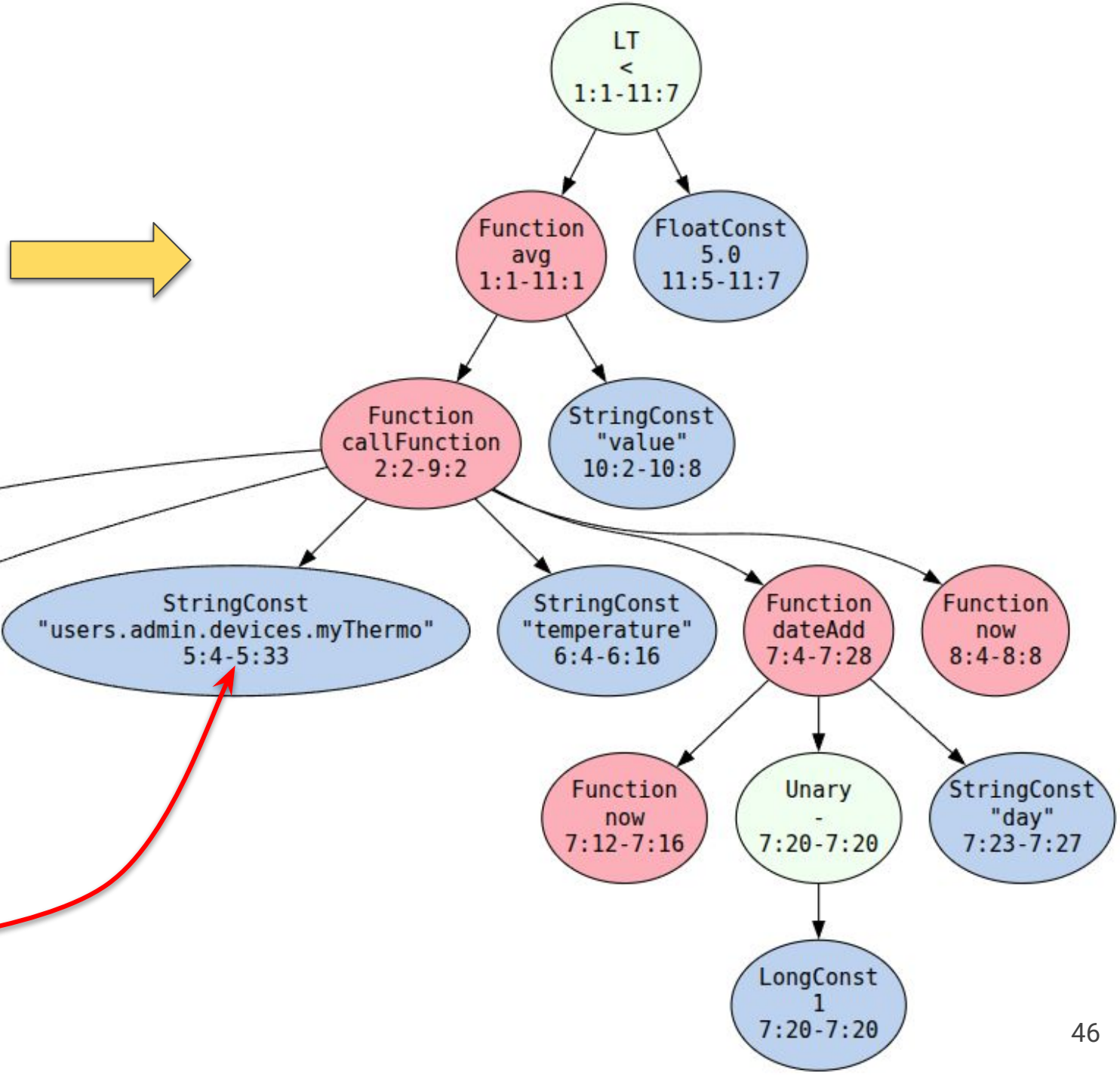
```
avg(
 callFunction(
   "utilities",
   "variableHistory",
   "users.admin.devices.myThermo",
   "temperature",
   dateAdd(now(), -1, "day"),
   now()
 ),
 "value"
) < 5.0
```




# НА ПРИМЕРЕ

выражения для напоминания
о переобувке машины

```
avg(
 callFunction(
   "utilities",
   "variableHistory",
   "users.admin.devices.myThermo",
   "temperature",
   dateAdd(now(), -1, "day"),
   now()
 ),
 "value"
) < 5.0
```




# НАВИГАЦИЯ

по номерам строк и столбцов
в исходном коде

# РАБОТАЕТ БЕЗУПРЕЧНО! (НЕТ)

- В языке есть вложенные выражения
  - Как аргумент функции **`eval()`** в JavaScript, Python, Ruby, …
- Или как аргумент ***условие*** у функции **СУММЕСЛИ** в Excel:

```
СУММЕСЛИ(диапазон; условие; [диапазон_суммирования])
```

```
СУММЕСЛИ(A2:A7, "Фрукты", C2:C7)
СУММЕСЛИ(A2:A5, ">1600")
СУММЕСЛИ(B2:B7, "*ция", C2:C7)
```

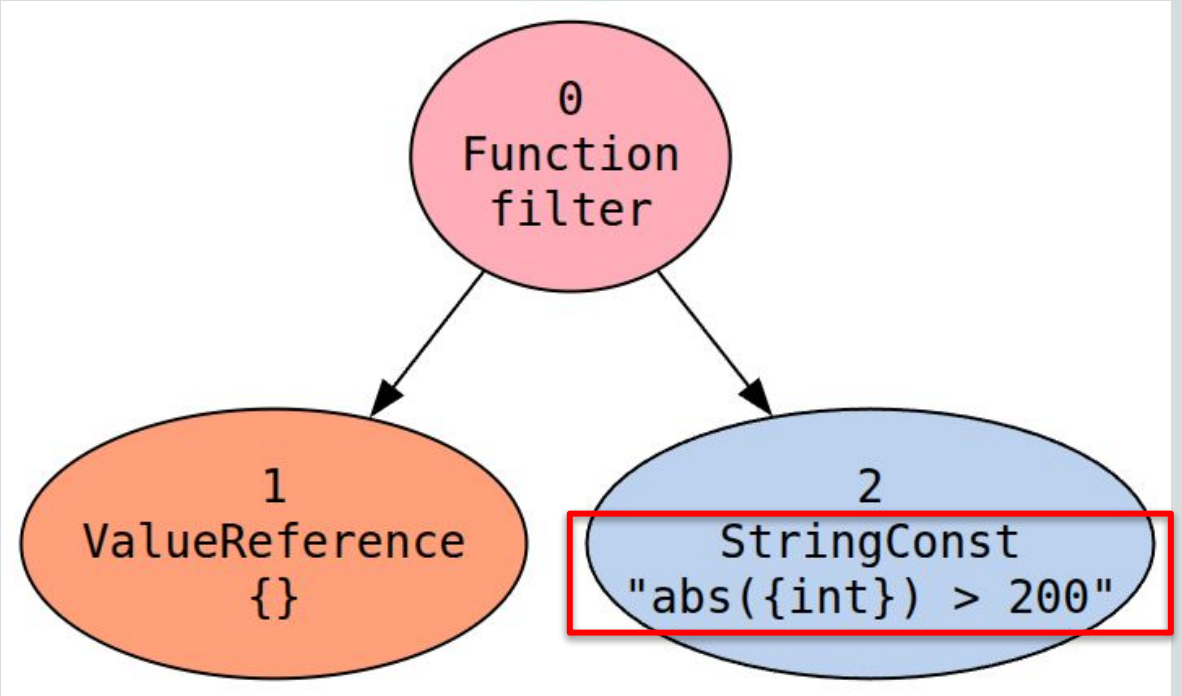
# ПРИМЕР: ФИЛЬТРАЦИЯ ТАБЛИЦЫ

**Выражение фильтрации**

```
filter({}, "abs({int}) > 200")
```

**Фильтруемая таблица**

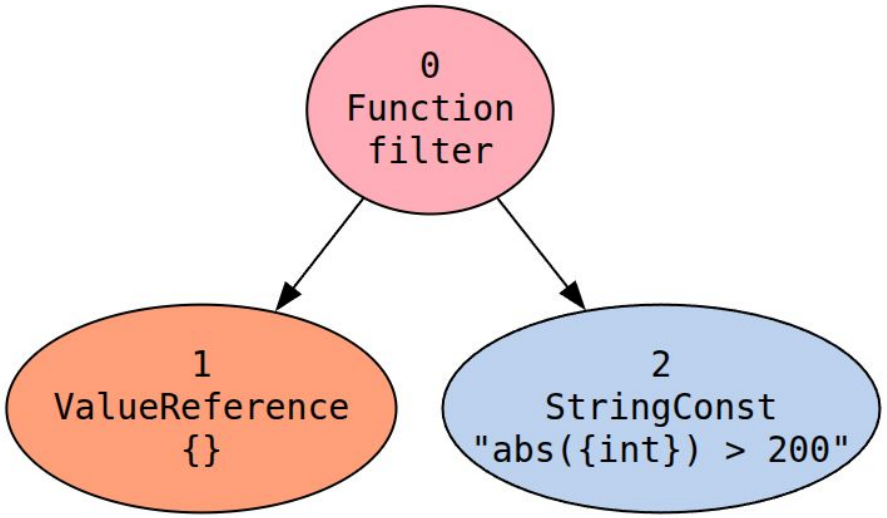
| str | int |
|---|---|
| test | 123 |
| **second** | **222** |
| **third** | **-333** |
| **fourth** | **444** |

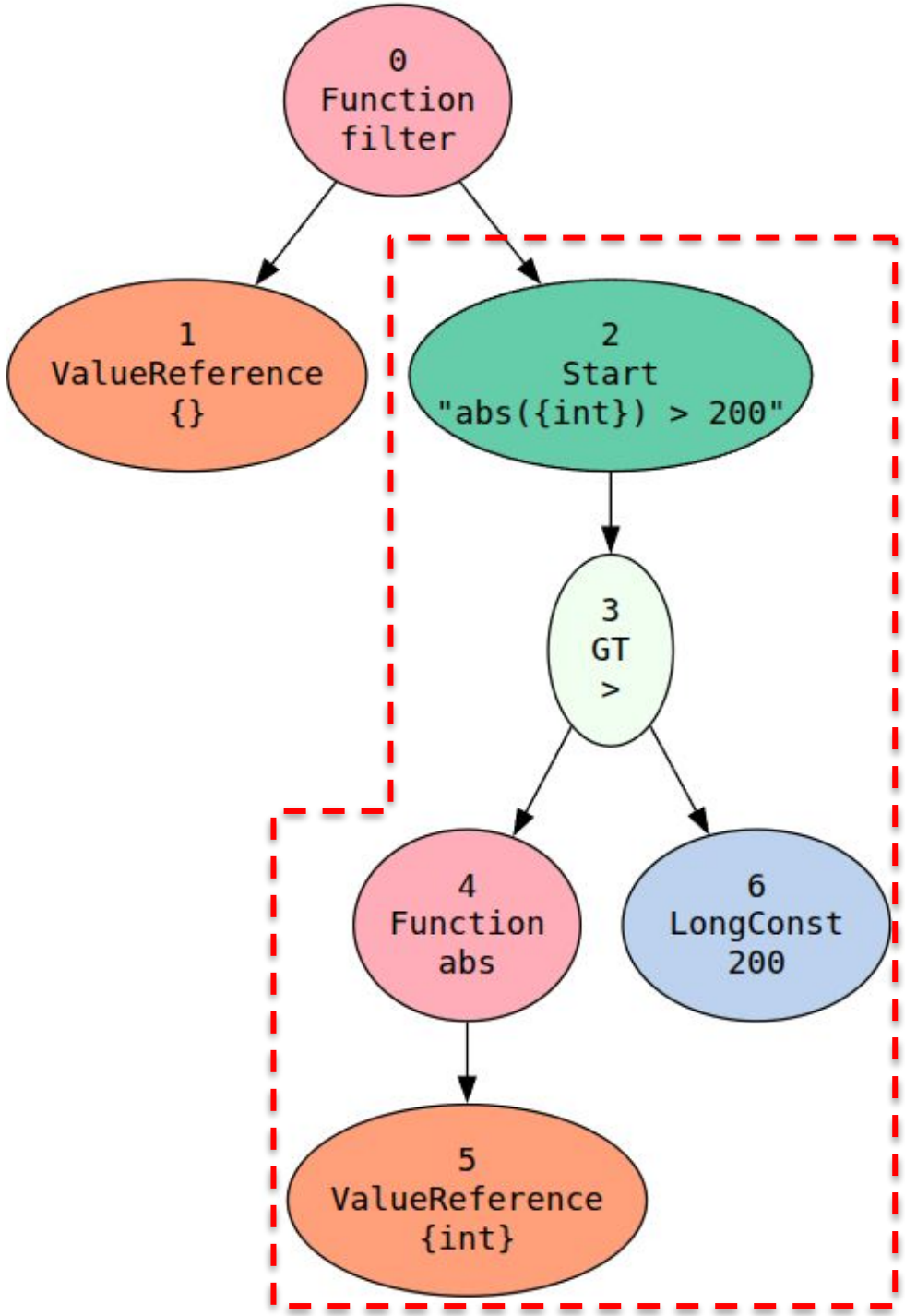


# ПРОБЛЕМЫ И РЕШЕНИЯ

- **Как отличить строку от выражения?**
  - Справочником для встроенных функций
  - По опциям для пользовательских функций
- **Как обозначить начало вложенного выражения в дереве?**
  - Синтетическим корневым узлом с типом `Start`
- **Как интегрировать одно дерево в другое?**
  - Подстановкой индекса обхода

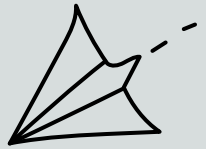
# КАК ЭТО ВЫГЛЯДИТ




`2:StringConst → 2:Start`

# КЭШИРОВАТЬ ВСЕ РАВНО НАДО



- **Наблюдения**:
  - выражения часто повторяются
  - динамических выражений может быть много тысяч
  - парсинг дешевый, но не бесплатный: время, GC pressure
- **Вывод**: деревья разбора нужно кэшировать
- **Ключ** кэширования – текст самого выражения

# ПОПУТНЫЕ ВЫВОДЫ

- При частом парсинге производительность решает
  - Поэтому даже JavaCC (или Congo) в деле
- Для отладки деревьев нужен простой формат их экспорта
  - Можно взять GraphViz, Gephi, PlantUML и т.п.
- Если парсинг частый, нужно кэшировать деревья разбора
  - И вовремя вытеснять: по старости, обращениям, весу и т.п.

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

**Сложно читать**

**Деревья разбора**


# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

| Сложно читать | Трудно отлаживать |
| --- | --- |
| Деревья разбора | Трассировка вычислений |
| 1 | 2 |

2

# ТРАССИРОВКА ВЫЧИСЛЕНИЙ

# В ЧЕМ ИДЕЯ

**Показать**

ход вычислений
с точностью до узла
дерева

# В ЧЕМ ИДЕЯ

**Показать**

ход вычислений
с точностью до узла
дерева

**Визуализировать**

с помощью тех же
деревьев разбора

# В ЧЕМ ИДЕЯ

**Показать**

ход вычислений
с точностью до узла
дерева

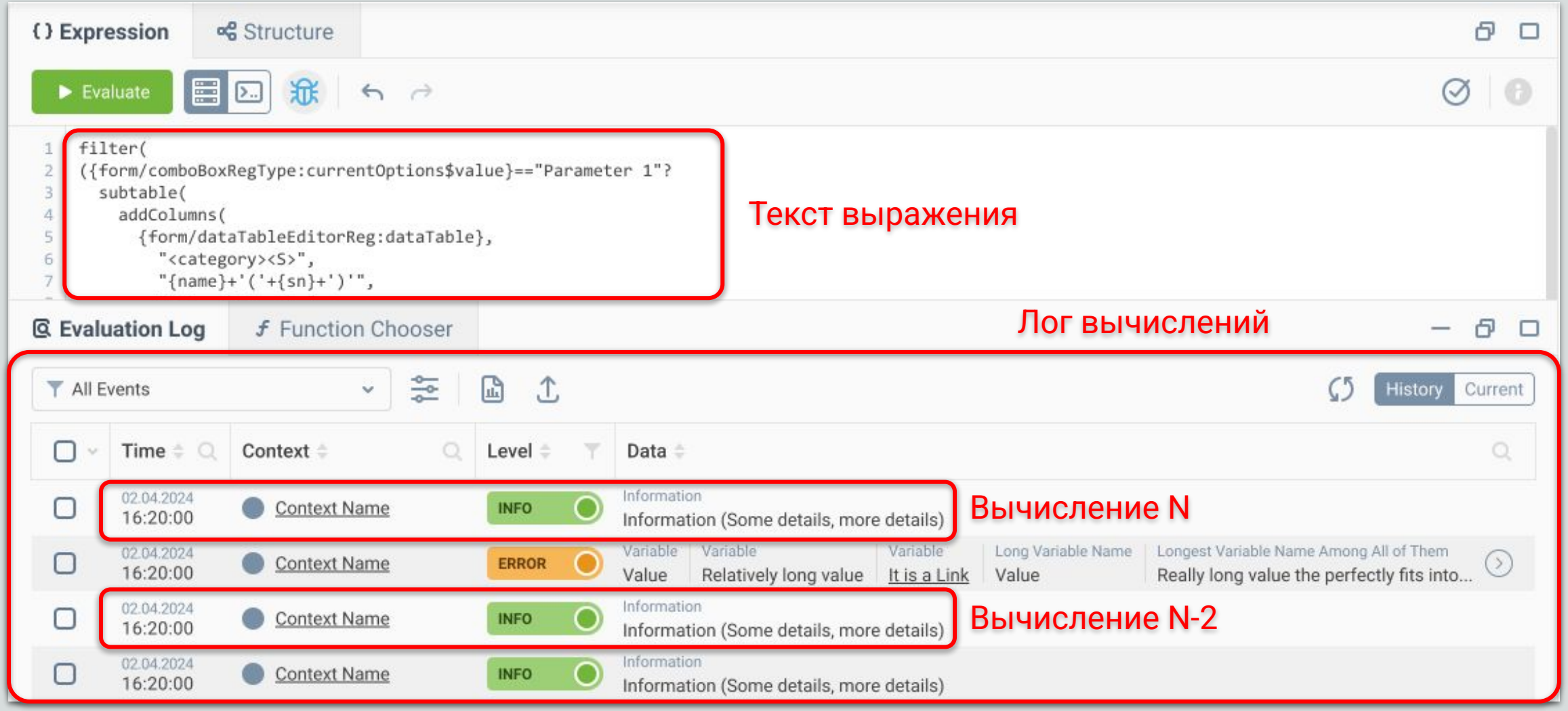
**Визуализировать**

с помощью тех же
деревьев разбора

**Обеспечить**

возможность легко
переключаться между
вычислениями

# КАК ЭТО ВЫГЛЯДИТ (ПРИМЕРНО)



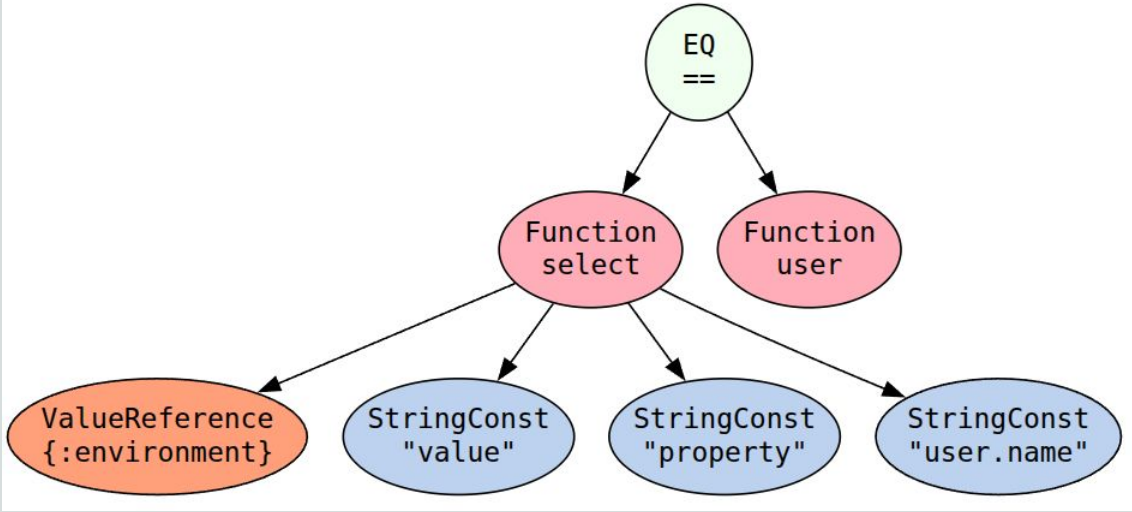
# ЗАЧЕМ: ПРИМЕР ИЗ ЖИЗНИ

**Сравнение имени пользователя в системных свойствах и логина в AggreGate**

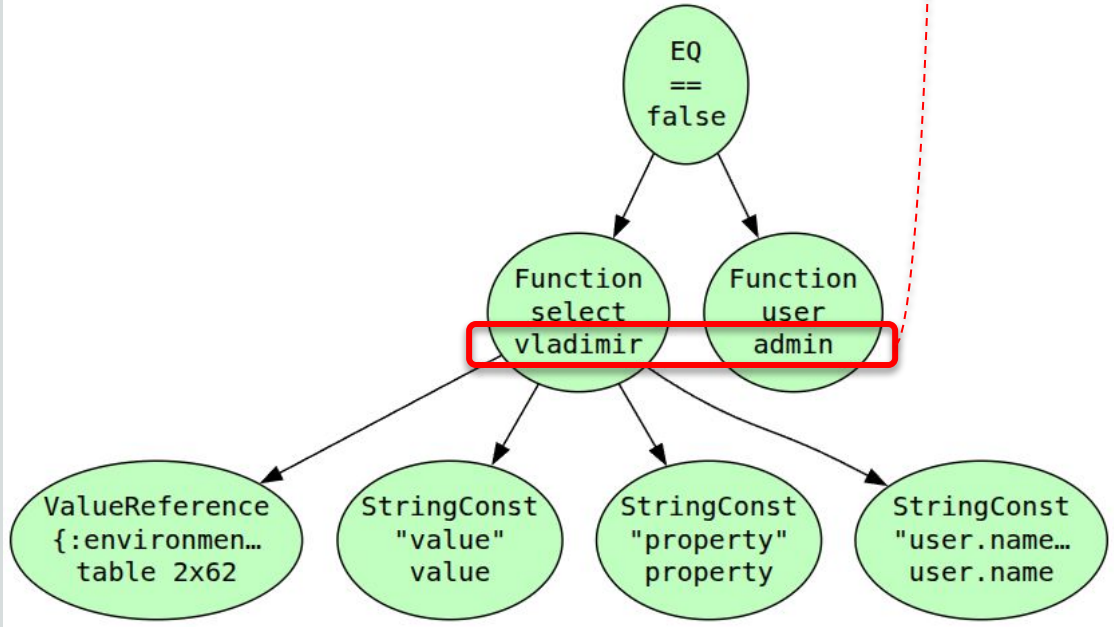
```
select({:environment}, "value", "property", "user.name") == user()
```

Почему-то выдает false 🤔

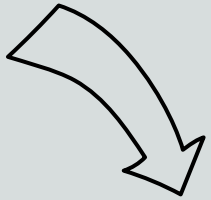
**Дерево разбора**




**Дерево вычисления**

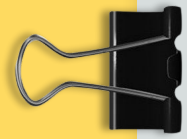


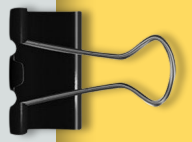
# ЖИЗНЕННЫЙ ЦИКЛ ВЫРАЖЕНИЯ



Вывод

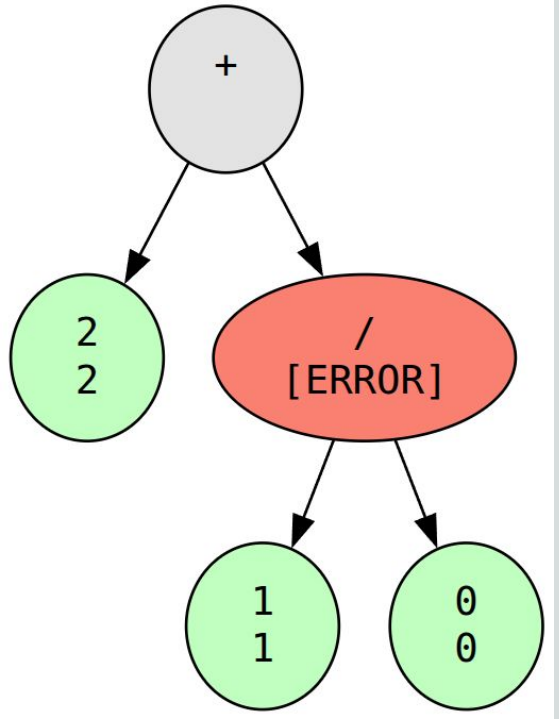
# Дерево разбора – подложка для множества вычислений





# ДЕТАЛЬНЫЕ ВЫВОДЫ

```
2 + 1 / 0
```

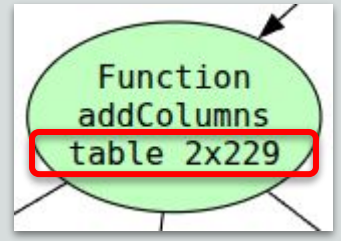



- Деревья вычислений ≠ деревья разбора
  - Вычислений сильно **больше** и они **чаще**
  - Вычисления могут охватывать **не все узлы** разбора
    - Например условия, &&, ||, **ошибки** и т.п.
- Но деревья нужно связывать для визуализации
  - **Индексы обхода** в помощь!
  - Основа – **дерево разбора**

# ВОПРОСЫ ПРОИЗВОДИТЕЛЬНОСТИ

- Выключенная трассировка не должна замедлять вычисления
  - Нужно уметь **в(ы)ключать** ее для отдельных выражений
- Доступ к недавним результатам должен быть мгновенным
  - Нужно **кэшировать** их в памяти
- Значения в узлах могут быть тяжелыми, но бесполезными
  - По умолчанию отдавать только их **строковые слепки**

*А как?* 🤔



# ТОЧКИ ПРИКРЕПЛЕНИЯ [PINPOINTS]

- Ссылки на “места дислокации” выражений, например:

**Выражение проверки в 1-ом триггере по переменной у тревоги thermoAlert**

```
{users.admin.alerts.thermoAlert:variableTriggers$expression[0]}
```

- Не зависят от текстов выражений
  - даже если тексты **повторяются**
- Уникальны в пределах сервера AggreGate

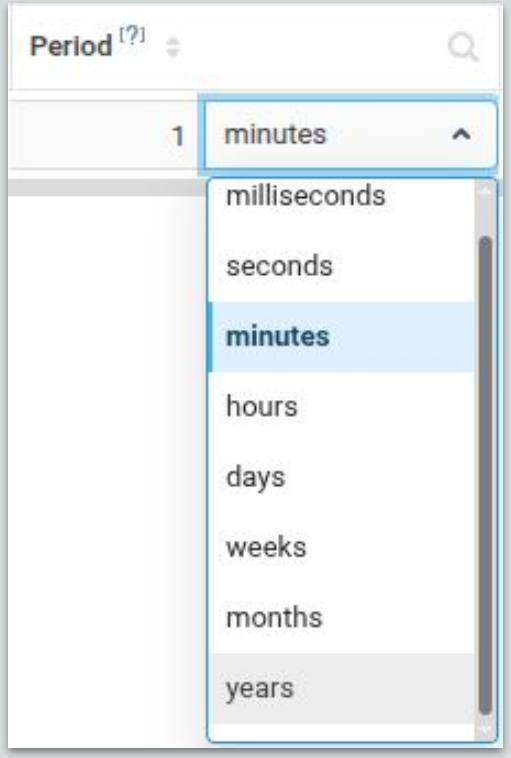
# ЛЕС В СБОРЕ

# РАБОТАЕТ БЕЗУПРЕЧНО! (НЕТ)

- Выражения могут вычисляться:
  - вручную (в редакторе)
  - по таймеру
  - по событию
  - по изменению переменных

*Это может быть очень часто!*

- Для вычислений построятся деревья результатов
  - и **положат** сервер...



# РЕШЕНИЕ: RATE LIMITER

- Прореживаем анализ результатов
  - с **настраиваемой** частотой
- Но не отбрасываем ручные вычисления
  - чтобы не испортить **troubleshooting**
- Предусматриваем мониторинг лимитера
  - чтобы помочь **диагностике**

Guava Library (Java)

https://static.insales-cdn.com/images/products/1/3527/516541895/Guava-Red-01.jpg

# ТЕПЕРЬ БЕЗУПРЕЧНО! (СНОВА НЕТ)

Сколько раз вычисляются узлы, если в выражении есть **вложенность**?

**Фильтрация таблицы по условию**

```
filter({}, "abs({int}) > 200")
```

1 раз

1 раз

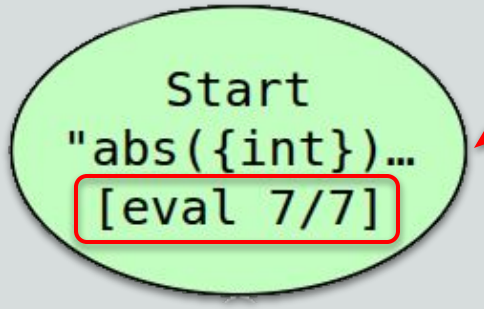
**N раз**
(по числу строк в таблице)

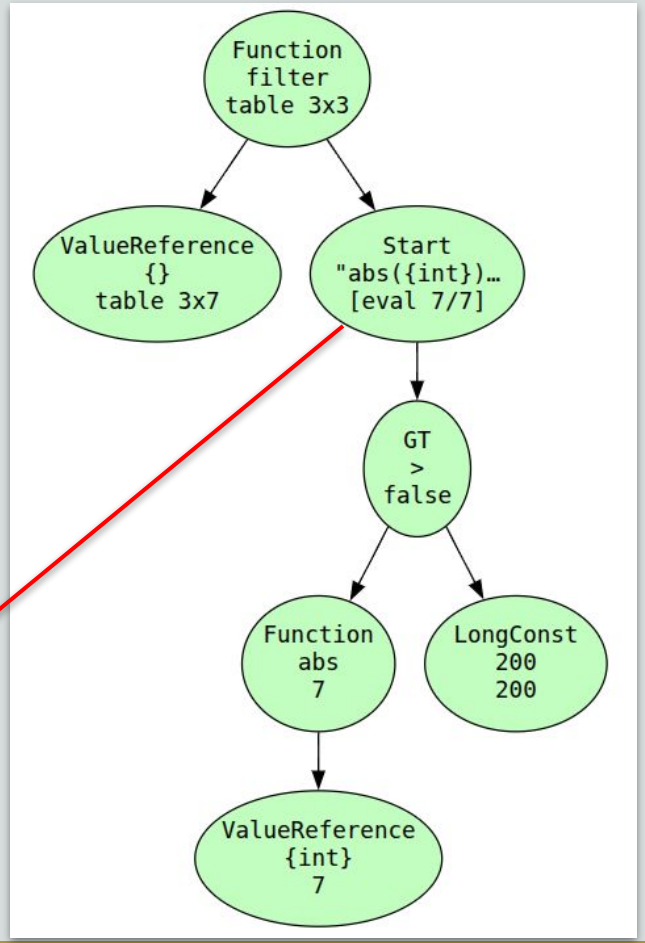
**Выводы**:

- Одному узлу разбора может соответствовать много узлов результата
- Узлы финальных деревьев могут ветвиться вглубь
- В узлах вложенных выражений нужен переключатель результатов

# РЕШЕНИЕ: МНОГОСЛОЙНЫЕ УЗЛЫ

- Для корней вложенных выражений учитываем:
  - **индекс** вычисления
  - **количество** вычислений
- В интерфейсе на таких узлах:
  - показываем стрелки-**переключатели**



# ПОПУТНЫЕ ВЫВОДЫ

- Вычисления должны храниться отдельно от парсинга
  - и лишь **накладываться** на него по запросу
- Объем вычислений нужно ограничивать
  - как через **кэш**, так и через **частоту** поступления
- Результаты отдельных узлов могут быть огромными
  - поэтому “**ленивизация**” API – наше всё

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

| Сложно читать | Трудно отлаживать |
| --- | --- |
| Деревья разбора | Трассировка вычислений |
| 1 | 2 |

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

| Сложно читать | Трудно отлаживать | Тяжело разбираться |
| --- | --- | --- |
| Деревья разбора | Трассировка вычислений | Граф взаимодействий |
| 1 | 2 | 3 |

3

# ГРАФ ВЗАИМОДЕЙСТВИЙ

# В ЧЕМ ИДЕЯ

**Выявить**

связи между
контекстами
в low-code-приложении

# В ЧЕМ ИДЕЯ

**Выявить**

связи между
контекстами
в low-code-приложении

**Описать**

их характер: тип,
частоту, длительность
и другие свойства

# В ЧЕМ ИДЕЯ

**Выявить**

связи между
контекстами
в low-code-приложении

**Описать**

их характер: тип,
частоту, длительность
и другие свойства

**Визуализировать**

в виде интерактивного
графа с возможностью
“погружения”

# ДВЕ НОВОСТИ ПРО ВЫЯВЛЕНИЕ

**Хорошая:**

- все **входящие** обращения локализованы в одном месте

**Так себе:**

- **исходящие** точки разбросаны по всей платформе

| **Следствие** |
| --- |
| Точки прикрепления во всех известных **исходящих** местах Платформы **пришлось** проставить **вручную** 🤦‍♂️ |

# КАК РЕАЛИЗОВАНО ОПИСАНИЕ

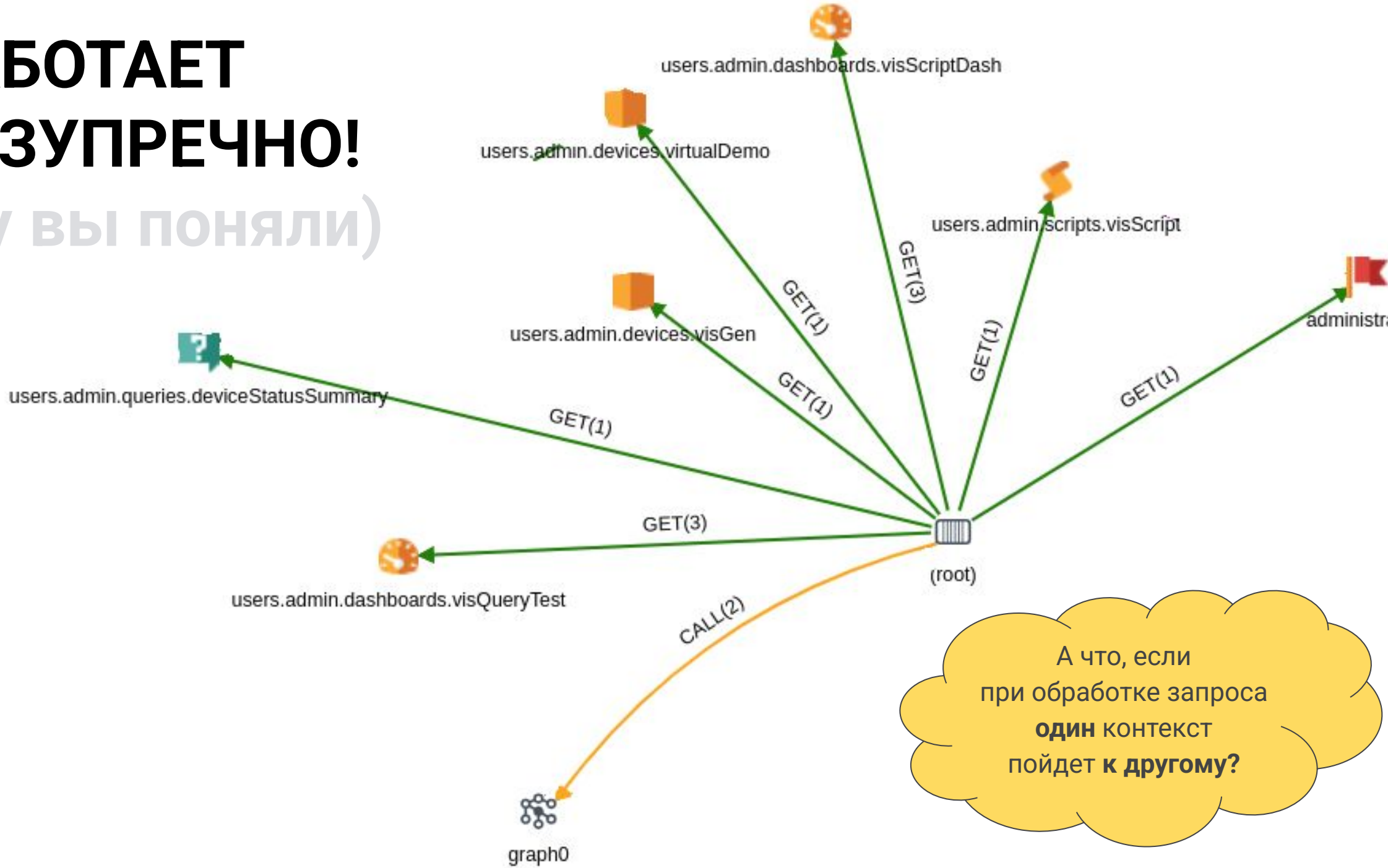
- **Типов** обращений всего **4**: GET/SET variable, FIRE event, CALL function
- Перехватчик обращений внедрен в **единое место их приема**
- И снабжен **подробностями** о pinpoint'ax:

**Пример описания исходящей точки**

```
Reference 'users.admin.devices.visGen:sine$value'
located at line 2, column 6
of expression originated from
users.admin.models.visModel:bindings$expression[0]
```

# РАБОТАЕТ БЕЗУПРЕЧНО!

(ну вы поняли)




А что, если
при обработке запроса
**один** контекст
пойдет **к другому?**

# ПОДДЕРЖКА ТРАНЗИТИВНОСТИ

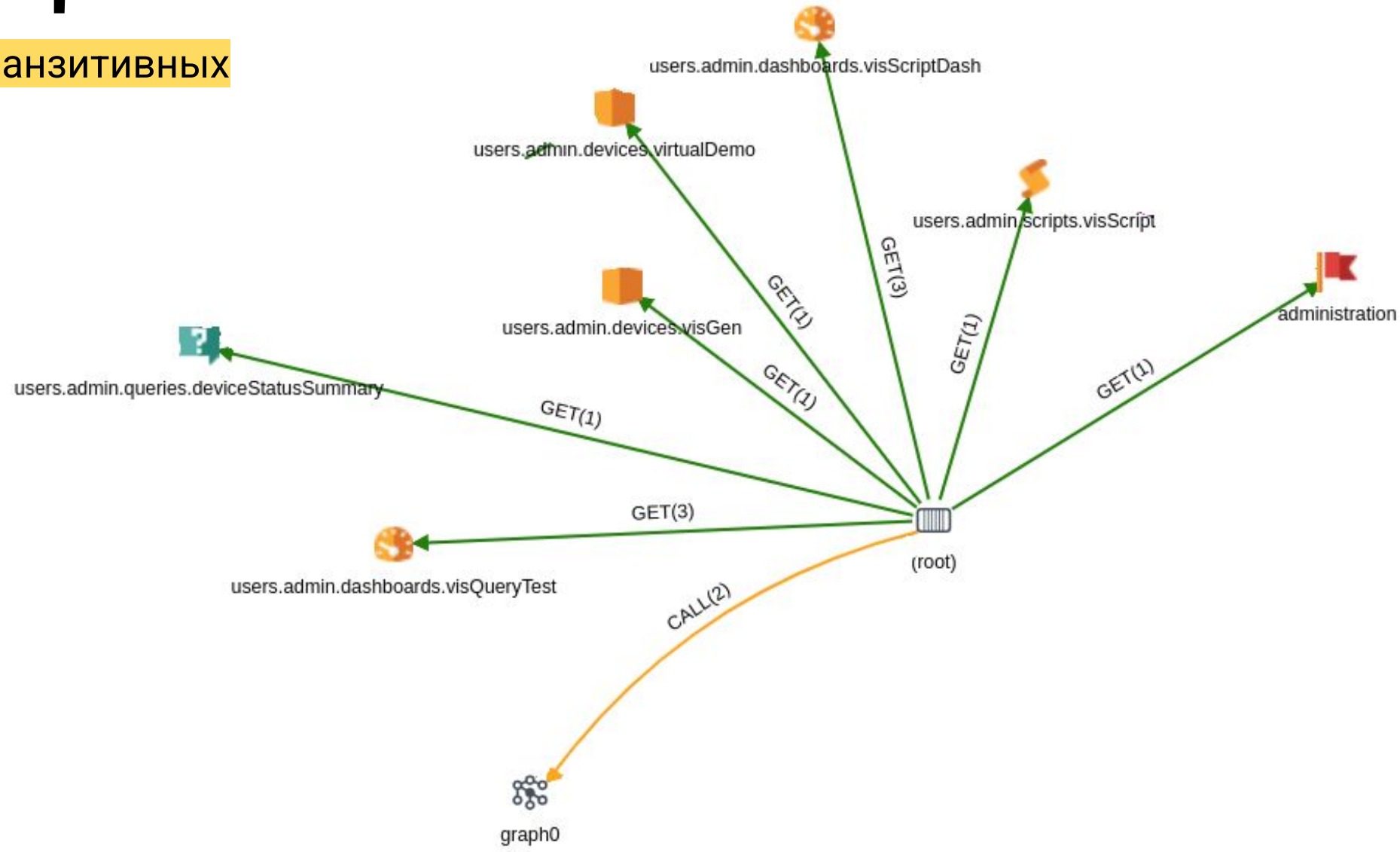
**Типичный пример транзитивного звена**

```
{:executeQuery("SELECT * FROM users.*:childInfo:status")}
```

- **Заводим** в каждом контексте поточно-локальный стек pinpoint'ов
- **Кладем** в него отправной pinpoint в начале обработки
- **Снимаем** со стека в конце обработки

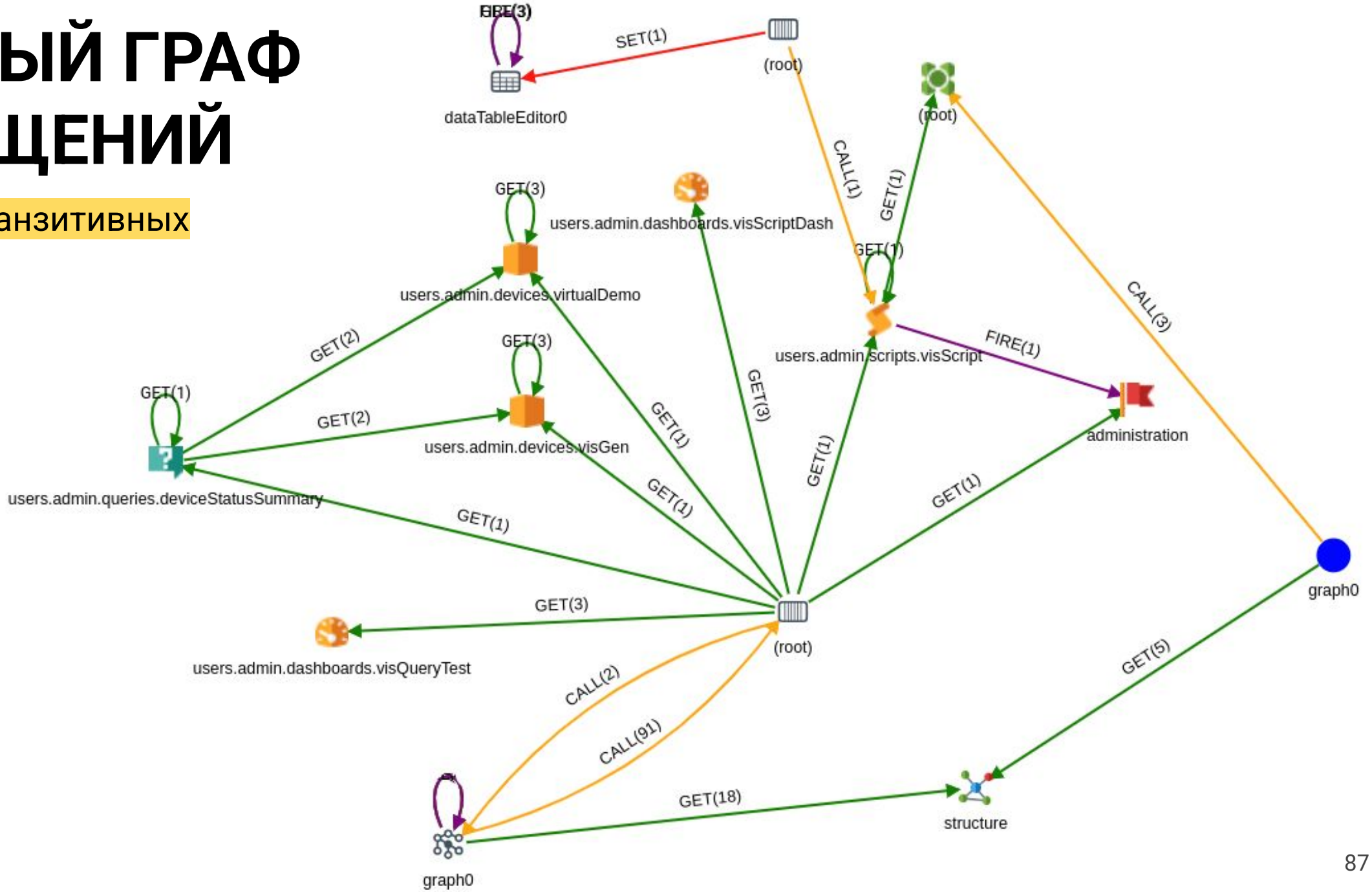
# ПОЛНЫЙ ГРАФ ОБРАЩЕНИЙ

С учетом транзитивных



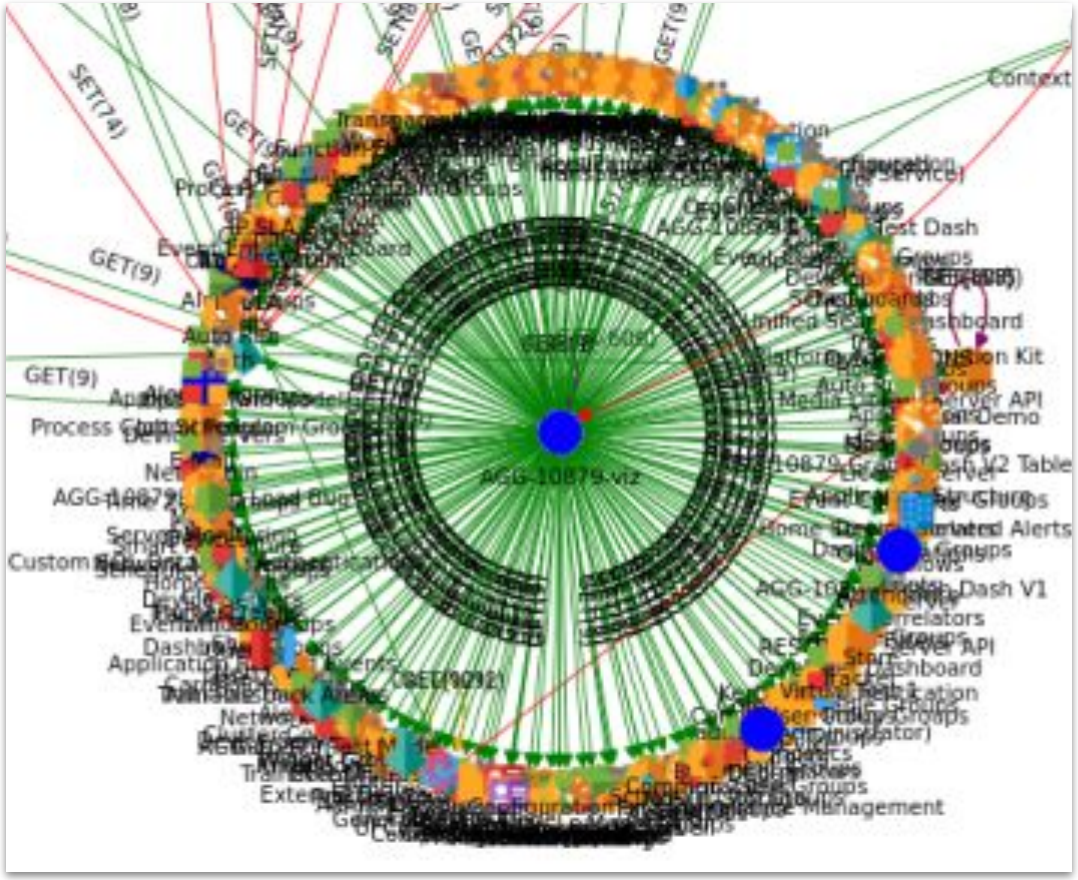
# ПОЛНЫЙ ГРАФ ОБРАЩЕНИЙ

С учетом транзитивных

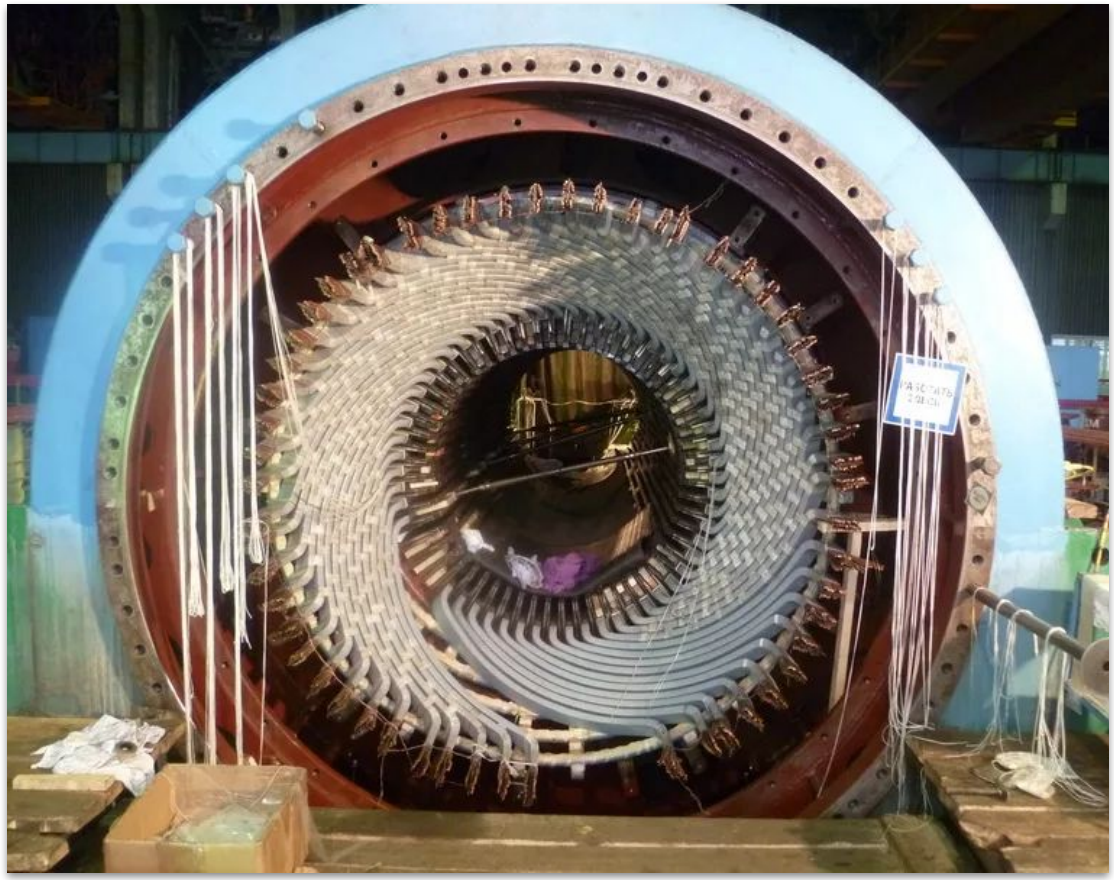


# НАЙДИТЕ 10 ОТЛИЧИЙ

Граф low-code-приложения на AggreGate




Обмотка турбогенератора ТВФ-60-2



Источник

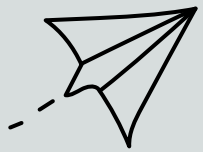
# ЧТО (НЕ) ТАК С ГРАФОМ?

**Хорошо** подходит для:

- описания общей структуры приложения
- выявления различных связей, в т.ч. транзитивных

**Плохо** подходит для:

- выяснения порядка и длительности взаимодействий
- выявления отдельных цепочек действий



# КАК МОЖНО УЛУЧШИТЬ

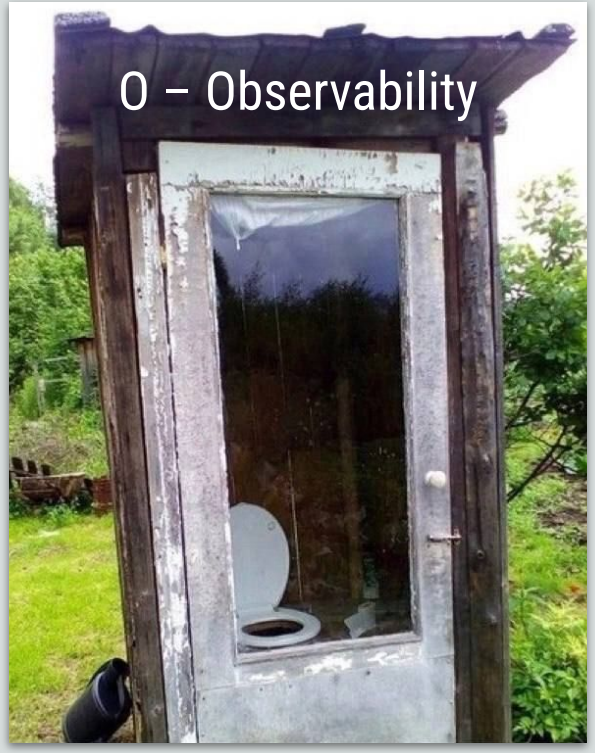
1. **Обозначить** границы действий
2. **Поддержать** для них вложенность

*Это ж **спаны**!*

3. **Научиться** собирать их в цепочки

   С уникальными ID

*А это **трейсы**!*

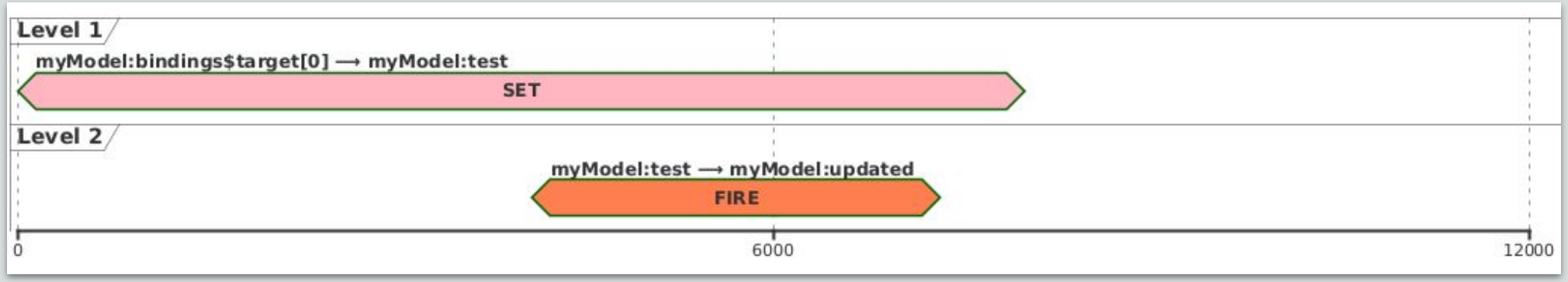


# OPEN TRACING НА МИНИМАЛКАХ

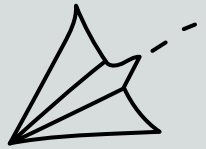
**Пример 2-звенной цепочки в формате PlantUML Timing Diagrams**

Уровень →
Начало →
Конец →

```
@L1
0 is SET #LightPink : myModel:bindings$target[0] → myModel:test
8000 is {hidden}
@L2
4089 is FIRE #Coral : myModel:test → myModel:updated
7330 is {hidden}
```



# ПОПУТНЫЕ ВЫВОДЫ

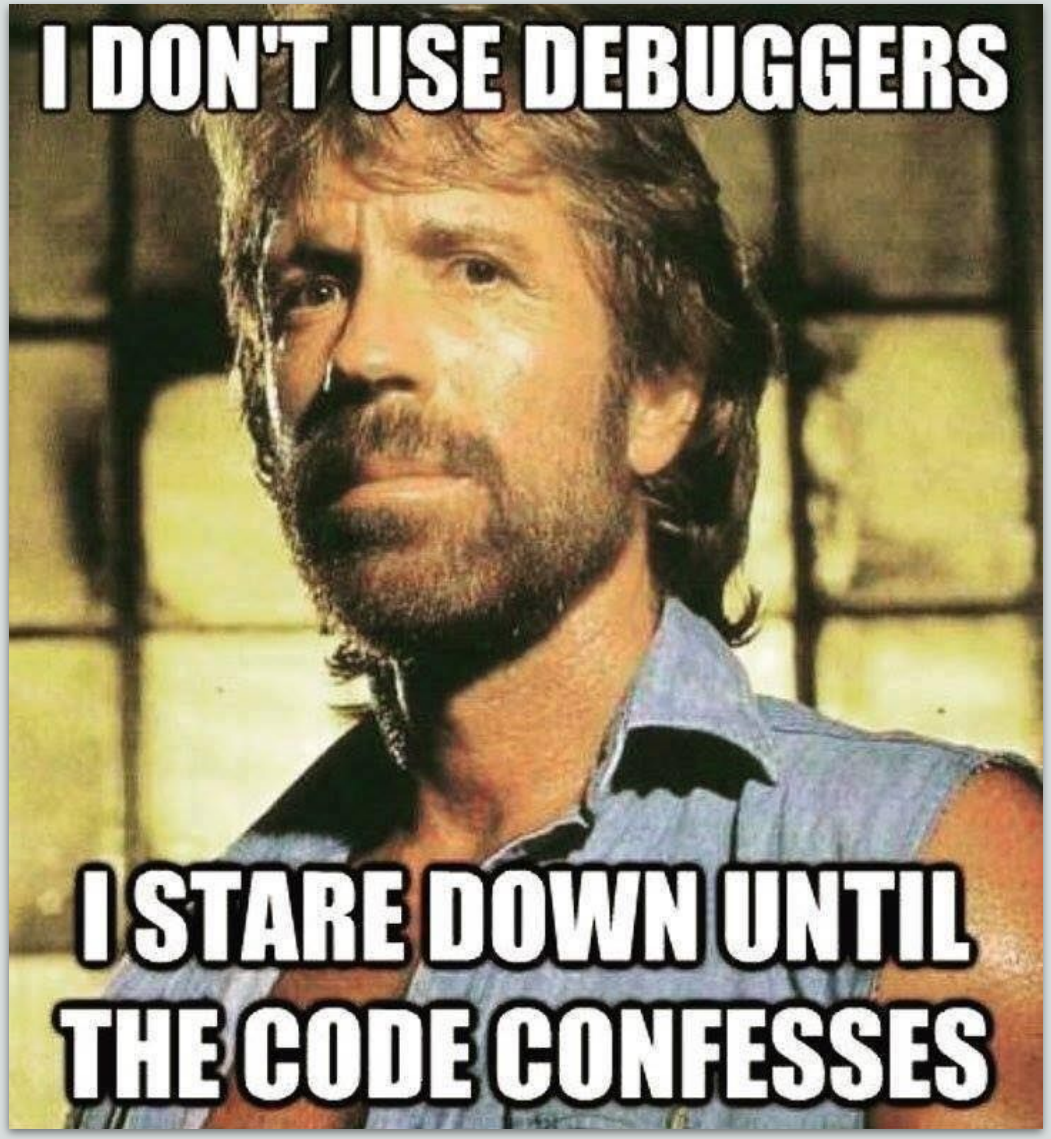


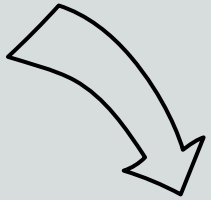
- Observability сама себя не обеспечит
  - Нужны **точки прикрепления**
- Графы структуры low-code похожи на деревья выражений
  - Главное отличие – **масштаб**
- Порядок и длительности действий лучше показывать отдельно
  - Идеи из **OpenTelemetry** вполне применимы

# ВЫБРАННЫЕ РЕШЕНИЯ ПРОБЛЕМ

| Сложно читать | Трудно отлаживать | Тяжело разбираться |
| --- | --- | --- |
| Деревья разбора | Трассировка вычислений | Граф взаимодействий |
| 1 | 2 | 3 |

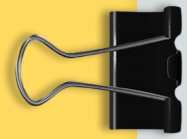
# LESSONS LEARNT

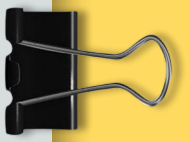


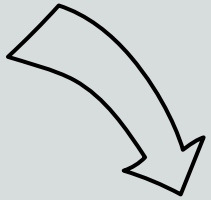


## 🧑‍🎨 Дизайнерам прикладных языков

# При выборе между простотой и гибкостью учитывайте отладку

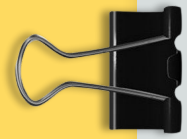


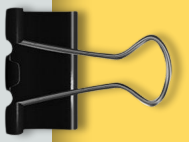




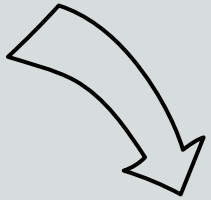
👩‍💻 Разработчикам парсеров/вычислителей

Заранее закладывайте
средства диагностики
для **всех*** сценариев

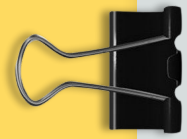




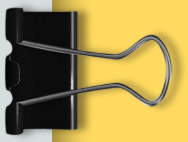
* прямых и ошибочных



👷 Архитекторам многомодульных систем



# Предотвращайте рост числа **типов** взаимодействия между модулями



# Как сделать исполнение low-code прозрачным: опыт большой IoT-платформы

**Владимир Плизга**
**(Tibbo Systems)**

@stopshelf
toparvion.pro

**Оцените доклад**

Saint HighLoad++ 2025

Генеральный партнер

garage eight